METHOD FOR DEFERRED DELETION OF ENTRIES FOR A DIRECTORY SERVICE BACKING STORE

BACKGROUND OF THE INVENTION

Technical Field

This invention relates generally to providing directory services in a distributed computing environment.

Description of the Related Art

A directory service is the central point where network services, security services and applications can form an integrated distributed computing environment. Typical uses of a directory services may be classified into several categories. A "naming service" (e.g., DNS and DCE Cell Directory Service (CDS)) uses the directory as a source to locate an Internet host address or the location of a given server. A "user registry" (e.g., Novell NDS) stores information about users in a system composed of a number of interconnected machines. The central repository of user information enables a system administrator to administer the distributed system as a single system image. Still another directory service is a "white pages" lookup provided by some e-mail clients, e.g., Netscape Communicator, Lotus Notes, Endora and the like).

With more and more applications and system services demanding a central information repository, the next generation directory service will need to provide system administrators with a data repository that can significantly ease administrative burdens. In addition, the future directory service must also provide end users with a rich information data warehouse that allows them to access department or company employee data, as well as resource information, such as name and location of printers, copy machines, and other environment resources. In the Internet/intranet environment, it will be required to provide user access to such information in a secure manner.

To this end, the Lightweight Directory Access Protocol (LDAP) has emerged as an IETF open standard to provide directory services to applications ranging from e-mail systems to distributed system management tools. LDAP is an evolving protocol that is based on a client-server model in which a client makes a TCP/IP connection to an LDAP server, sends requests, and receives responses. The LDAP information model in particular is based on an "entry," which contains information about some object. Entries are typically organized in a specified tree structure, and each entry is composed of attributes.

LDAP provides a number of known functions including query (search and compare), update, authentication and others. The search and compare operations are used to retrieve information from the database. For the search function, the criteria of the search is specified in a search filter. The search filter typically is a Boolean expression that consists of qualifiers including attribute name, attribute value and Boolean operators like AND, OR and NOT. Users can use the filter to perform complex search operations. One filter syntax is defined in RFC 2254.

LDAP thus provides the capability for directory information to be efficiently queried or updated. It offers a rich set of searching capabilities with which users can put together complex queries to get desired information from a backing store. Increasingly, it has become desirable to use a relational database for storing LDAP directory data. Representative database implementations include DB/2, Oracle, Sybase, Informix and the like. As is well known, Structured Query Language (SQL) is the standard language used to access such databases.

In implementing an LDAP directory service with a relational database backing store, deleting an entry from the directory involves deleting rows from several different tables. In particular, in addition to the LDAP entry table, which stores an entry ID, parent ID,

create and last modified times, together with the complete entry in string format, the schema includes a separate table for each attribute. When an entry is to be deleted, a global lock is placed on all of these tables (including the entry table and its associated attribute tables) until the delete is processed. As a result, all other query activity into the database is locked out for whatever time period is required for the backing store to return an indication that the delete operation has been completed. This is a very time consuming and computationally-intensive process.

BRIEF SUMMARY OF THE INVENTION

It is a primary object of this invention to reduce the time required to perform a delete operation in a directory service having a relational database backing store.

It is another object of the present invention to delete an entry from a directory without having to lock out all other query activity during the operation as is presently required by the prior art.

A further object of the invention is to reduce the apparent processing time required to delete an entry from a directory by deferring the actual deletion until execution of a cleanup handler thread.

It is thus object of the present invention to provide a simple and efficient technique for speeding up entry deletion by deferring the actual deletion of rows from a database, preferably until invocation of a cleanup routine.

A specific object of this invention is to provide a more efficient LDAP directory service having a relational database management system (DBMS) as a backing store.

A general object of this invention is to provide a reliable and scaleable enterprise directory solution, wherein a preferred implementation is LDAP using a DB/2 backing store.

The present invention overcomes the deficiencies of the prior art. When an entry is to be deleted, its entry in an entry table (e.g., the ldap_entry table) is tagged deleted, preferably by setting its creation time to a given value (e.g., a null value). This operation involves a change to only a single, unindexed field in a single row in a single table and, as a result, is quite efficient. At periodic intervals, a cleanup thread performs actual row deletions for any entry tagged as deleted. When searches are done in the directory, the invention preferably modifies the SQL query to exclude rows with a null change time, thus preventing deleted entries from being returned by the search.

In a preferred embodiment, a method for deleting entries from a directory in which directory information is stored in a set of database tables begins upon a request to delete a directory entry. In response, the directory entry is tagged, preferably by setting the entry's creation time to a null value. If a search query is received thereafter, the method excludes tagged entries from search results that would otherwise satisfy the search query. At a periodic interval, the routine searches for tagged entries, and references to the tagged entries are then deleted throughout the set of database tables. Thus, the inventive method defers entry deletions to enable directory queries to be

processed even if deleted entries have not yet been fully expunged from the directory.

The foregoing has outlined some of the more pertinent objects and features of the present invention. These objects and features should be construed to be merely illustrative of some of the more prominent features and applications of the invention. Many other beneficial results can be attained by applying the disclosed invention in a different manner or modifying the invention as will be described. Accordingly, other objects and a fuller understanding of the invention may be had by referring to the following Detailed Description of the preferred embodiment.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention and the advantages thereof, reference should be made to the following Detailed Description taken in connection with the accompanying drawings in which:

Figure 1 is a representative LDAP directory service implementation;

Figure 2 is a simplified LDAP directory;

Figure 3 is a flowchart of an LDAP directory session;

Figures 4A-4B show representative LDAP directory service implementations having relational database backing store;

Figure 5 is a representative LDAP entry table prior to being modified in accordance with the present invention;

Figure 6 is the LDAP entry table of Figure 5 after being modified to mark a given entry as deleted in accordance with the invention; and

Figure 7 is a simplified flowchart of the inventive cleanup handler routine for deleting tagged entries in the entry and attribute tables.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A block diagram of a representative LDAP directory service in which the present invention may be implemented is shown in **Figure 1**. As is well-known, LDAP is the lightweight directory access protocol, and this protocol has been implemented in the prior art, e.g., as either a front end to the X.500 directory service, or as a standalone directory service. According to the protocol, a client machine **10** makes a TCP/IP connection to an LDAP server **12**, sends requests and receives responses. LDAP server **12** supports a directory **21** as illustrated in a simplified form in **Figure 2**. Each of the client and server machines further include a directory runtime component **25** for implementing the directory service operations as will be described below. The directory **21** is based on the concept of an "entry" **27**, which contains information about some object (e.g., a person). Entries are composed of attributes **29**, which have a type and one or more values. Each attribute **29** has a particular syntax that determines what kinds of values are allowed in the attribute (e.g., ASCII characters, .jpeg file, etc.) and how these values are constrained during a particular directory operation. Entries are stored in a given table (e.g., ldap_entry) that includes entry IDs, parent IDs, a create and last modified timestamp, together with the complete entry in string format. Each attribute has an associated attribute table.

The directory tree is organized in a predetermined manner, with each entry uniquely named relative to its sibling entries by a "relative distinguished name" (RDN). An RDN comprises at least one distinguished attribute value from the entry and, at most, one value from each attribute is used in the RDN. According to the protocol, a globally unique name for an entry, referred to as a "distinguished name" (DN), comprises a concatenation of the RDN sequence from a given entry to the tree root.

The LDAP search can be applied to a single entry (a base level search), an entry's children (a one level search), or an entire subtree (a subtree search). Thus, the scope supported by LDAP search are: base, one level and subtree. LDAP does not support search for arbitrary tree levels and path enumeration.

LDAP includes an application programming interface (API), as described in "The C LDAP Application Program Interface", IETF Working Draft, July 29, 1997, which is incorporated herein by reference. An application on a given client machine uses the LDAP API to effect a directory service "session" according to the flowchart of **Figure 3.** At step **40**, an LDAP session with a default LDAP server is initialized. At step **42**, an API function ldap_init() returns a handle to the client, and this handle may allow multiple connections to be open at one time. At step **44**, the client authenticates to the LDAP server using, for example, an API ldap_bind() function. At step **46**, one or more LDAP operations are performed. For example, the API function ldap_search() may be used to perform a given directory search. At step **48**, the LDAP server returns the results of the directory search, e.g., one or more database elements that meet the search criteria. The session is then closed at step **50** with the API ldap_unbind() function then being used to close the connection.

It may be desirable to store LDAP directory data in a backing store. **Figures 4A-4B** illustrates several representative LDAP directory service implementations that use a relational database management system (RDBMS) for this purpose. These systems merely illustrate possible LDAP directory services in which the present invention may be implemented. One of ordinary skill should appreciate, however, that the invention is not limited to an LDAP directory service provided with a DB/2 backing store. The principles of the present invention may be practiced in other types of directory services (e.g., X.500) and using other relational database management systems (e.g., Oracle, Sybase, Informix, and the like) as the backing store.

In **Figure 4A**, an LDAP client **34** can connect to a number of networked databases **38**a-**58**n through an LDAP server **36**. The databases **38**a-**38**n contain the directory information. However, from the user's perspective, the LDAP server **36** stores all the information without knowing the database **38** in which the data is actually located. With this configuration, the LDAP server **36** is freed from managing the physical data storage and is able to retrieve information from multiple database servers **38** which work together to form a huge data storage.

Figure 4B illustrates a multiple client/multiple server LDAP/DB2 enterprise solution. In this environment, a DB/2 client preferably runs on each LDAP server **36**. Each such DB/2 client can connect to any database server **38** containing directory information. The collection of database servers **38**a-**38**n form a single directory system image, and one or more of the LDAP servers **36** can access such information. Because all the LDAP servers **36** see the same directory image, a network dispatcher **37** may be deployed to route requests among the LDAP servers **36**.

One of ordinary skill should appreciate that the system architectures illustrated in **Figures 4A-4B** are not to be taken as limiting the present invention. The inventive technique may be used to search any relational database using hierarchical, filter-based database queries.

The present invention is a method for deferred deletion of entries in a directory service backing store. In LDAP, an entry is deleted using an SQL statement. In the prior art, the directory server responds to the delete entry statement by instituting a global lock on the database tables to ensure that data in those tables cannot be modified while the entry is being deleted from the directory. The present invention provides an enhanced delete operation whereby the entry is simply marked for deletion at a later time. This deferred entry deletion routine is now described in more detail below.

By way of brief background, **Figure 5** illustrates a representative portion of the entry table **60** (e.g., ldap_entry). This table comprises a number of columns including the entry identifier (EID) **62**, the parent identifier (PEID) **64**, the owner **66**, the creation date **68**, the last modified date **70**, the actual entry data **72** (in a string format), the entry size **74**, and additional information **76**. In this example, each of the entries has a given creation date. According to the invention, when a user requests deletion of an entry, the creation date for that entry is set to a given null value. **Figure 6** illustrates this operation after the user has requested deletion of entry 4. As can be seen, the null value is present in the creation data column **68**, and the timestamp for the last modified date **70** is updated to reflect when the delete request was received. At this point, the entry is conditionally deleted, even though the data for this entry remains in the entry table (as well as in the attribute tables).

Thus, according to the invention, when an entry is to be deleted, its entry in an entry table is tagged as deleted, preferably by setting its creation time to a given value (e.g., a null value). As illustrated in **Figures 5-6**, this operation involves a change to only a single, unindexed field in a single row in a single table and, as a result, is quite efficient. At periodic intervals, a cleanup thread is then used to perform actual row deletions for any entry tagged as deleted. This operation is illustrated in the flowchart of **Figure 7**. The thread begins at step **82** to test whether a given time period has expired. If not, the thread cycles. If, however, the time period has expired, the thread continues at step **84** to start the cleanup thread. At step **86**, the thread searches the LDAP table to identify records marked for deletion. The routine then continues at step **88** to test whether all entries have been processed. If so, the routine routines to step **82**. If, however, the routine has not processed all marked entries, the routine continues at step **90** to get the next marked entry. At step **92**, the routine deletes the record from the entry table. At step **94**, the routine deletes all associated records in the attribute tables. Control then returns back to step **88** and the process repeats as needed.

Because entries are merely marked for deletion, the present invention also includes a routine for processing directory search queries into the modified entry table. Thus, as compared to the prior art, the present invention enables a user to perform search and other queries into the directory despite the existence of the entries tagged for deletion. This operation preferably is achieved by modifying the SQL statements to exclude rows with a null change time, thus preventing deleted entries from being returned by the search. This operation is illustrated below.

By way of brief background, the following is a representative SQL query searching for sn = Bachmann:

SELECT distinct EDIR.LDAP_entry.EID from EDIR.LDAP_entry, EDIR.LDAP-DESC where (EDIR.LDAP_entry.EID = EDIR.LDAP_DESC.DEID and EDIR.LDAP_DESC.AEID = 6142) and EDIR.LDAP_entry.EID in (select EID from EDIR.SN where SN = BACHMANN)).

According to the present invention, the above query is modified to ignore entries marked as deleted, preferably as follows:

SELECT distinct EDIR.LDAP_entry.EID from EDIR.LDAP_entry, EDIR.LDAP_DESC, where (EDIR.LDAP_entry.EID = EDIR.LDAP_DESC.DEID and EDIR.LDAP.DESC.AEID = 6142) and EDIR.LDAP_entry.EID in (Select EID from EDIR.SN where SN = Bachmann) and EDIR.LDAP_entry.Create.Timestamp <> Ø).

As can be seen, the last clause of the SQL statement looks for entries that have their creation timestamp as non-zero. This operation prevents deleted entries from being returned by the search.

As previously described, at periodic intervals, the routine tests to determine which records have been marked for deletion. This was step **76** in **Figure 7**. A representative SQL query to find records marked for deletion is then as follows:

Select distinct EDIR.LDAP_entry EID from EDIR.LDAP_entry, where Create_Timestamp = Ø.

Thus, according to a preferred embodiment, a method for deleting entries from a directory in which directory information is stored in a set of database tables begins upon a request to delete a directory entry. In response, the directory entry is tagged, preferably by setting the entry's creation time to a null value. If a search query is received thereafter, the method excludes tagged entries from search results that would otherwise satisfy the search query. At a periodic interval, the routine then periodically searches for tagged entries, and references to the tagged entries are then deleted throughout the set of database tables. Thus, the inventive method defers entry deletions to enable directory queries to be processed even if deleted entries have not yet been fully expunged from the directory.

The inventive scheme preferably takes advantage of several LDAP table structures that are now described below.

Entry table:

This table holds the information about a LDAP entry. This table is used for obtaining the EID of the entry and supporting LDAP_SCOPE_ONELEVEL and LDAP_SCOPE_BASE search scope.

-EID. The unique identifier of the LDAP entry. This field is indexed.

-PEID. The unique identifier of a parent LDAP entry in the naming hierarchy.

-EntryData. Entries are stored using a simple text format of the form attribute: value. Non-ASCII values or values that are too long to fit on a reasonable sized line are represented using a base 64 encoding. Giving an ID, the corresponding entry can be returned with a single SELECT statement.

Descendant table:

The purpose of this table is to support the subtree search feature of LDAP. For each LDAP entry with an unique ID (AEID), this table contains the descendant entries unique identifiers (DEID). The columns in this table are:

-AEID. The unique identifier of the ancestor LDAP entry. This entry is indexed.

-DEID. The unique identifier of the descend LDAP entry. This entry is indexed.

Attribute table:

One table per searchable attribute. Each LDAP entry is assigned an unique identifier (EID) by the backing store. The columns for this table are:

-EID

-Attribute value

Thus, in the parent table, the EID field is the unique identifier of an entry in the LDAP naming hierarchy. The PEID field is the unique identifier of the parent entry in the naming hierarchy. In the descendant table, the AEID field is the unique identifier of a ancestor LDAP entry in the LDAP naming hierarchy. The DEID field is the unique identifier of the descend LDAP entry.

In addition to the table structures described above, the following SQL SELECT statements are used by LDAP/DB2 search routines:

Base Level Search:

```
SELECT entry.EntryData,
    from ldap_entry as entry
        where entry.EID in (

select distinct ldap_entry.EID
from -table list-
where (ldap_entry.EID=-root dn id- )
 -sql where expressions-)
```

One Level Search:

```
SELECT entry.EntryData,
    from ldap_entry as entry
```

where distinct
ldap_entry.EID
from ldap_entry, -table
list-
ldap_entry as pchild, -list of tables-
where ldap_entry.EID=pchild.EID
AND pchild.PIED=-root dn id-
-sql where expressions-)
Subtree Search
SELECT entry.EntryData,
from ldap_entry as entry
where entry.EID in (
select distinct ldap_entry.EID
from ldap_entry, ldap_desc, -table list-
where
(LDAP_ENTRY.EID=ldap_desc.DEID AND
ldap_desc.AEID=-root dn id-)
ldap_entry as pchild. -table list-
where ldap_entry.EID=ldap_desc.EID
AND ldap_desc.AEID=%d -where expressions-).

In the above representation, -table list- and -where expression- are the two null terminated strings returned by the SQL generator. The -root dn id- is the unique identifier of the root dn. The where clause should only be generated if -where expression- is not the empty string and no errors where detected in the parsing the LDAP filter.

As noted above, the invention may be implemented in any hierarchical directory service in which a relational database management system (RDBMS) is used to provide a backing store function. Thus, for example, the principles of the invention may be carried out in an X.500 directory service or hereinafter-developed LDAP implementations. The SQL query generated according to the present invention is used to access the relational database, and results are then returned in response to this query. The invention may also be implemented within a relational database management system being used as an add-on to a directory service. One of ordinary skill will appreciate that the invention can be applied to any relational database management system (RDBMS) and not simply DB/2, the implementation described above. Thus, for example, the relational database may be Oracle, Sybase or any other third party supplied backing store. In addition, the EID sets approach can also be applied to b-tree based LDAP server implementation.

Moreover, although the preferred embodiment has been described in the context of deleting an LDAP entry in a relational database backing store, the inventive technique should be broadly construed to extend to deleting entries from any type of directory in which directory information is stored in a set of database tables. Thus, the present invention is not limited to use with hierarchical directories. Rather, as noted above, the techniques described herein may be implemented in conjunction with any higher level directory structure in which information is spread out over a set of tables.

One of the preferred embodiments of the routine of this invention is as a set of instructions (e.g., computer program code) in a code module resident in or downloadable to the random access memory of a computer. Until required by the computer, the set of instructions may be stored in another computer memory, for example, in a hard disk drive, or in a removable memory such as an optical disk (for eventual use in a CD ROM) or floppy disk (for eventual use in a floppy disk drive), or downloaded via the Internet or other computer network.

In addition, although the various methods described are conveniently implemented in a general purpose computer selectively activated or reconfigured by software, one of ordinary skill in the art would also recognize that such methods may be carried out in hardware, in firmware, or in more specialized apparatus constructed to perform the required method steps.

Having thus described our invention, what we claim as new and desire to secure by Letters Patent is set forth in the following claims.